Multifamily Building Stock Modeling

Cooperative Research and Development Final Report

CRADA Number: CRD-17-00701

NREL Technical Contact: Eric Wilson

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Contract No. DE-AC36-08GO28308

Technical Report
NREL/TP-5500-73294
June 2023

Multifamily Building Stock Modeling

Cooperative Research and Development Final Report

CRADA Number: CRD-17-00701

NREL Technical Contact: Eric Wilson

Suggested Citation

Wilson, Eric. 2023. *Multifamily Building Stock Modeling: Cooperative Research and Development Final Report, CRADA Number CRD-17-00701.* Golden, CO: National Renewable Energy Laboratory. NREL/TP-5500-73294. https://www.nrel.gov/docs/fy23osti/73294.pdf.

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Contract No. DE-AC36-08GO28308

Technical Report
NREL/TP-5500-73294
June 2023

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

NOTICE

This work was authored [in part] by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Building Technologies Office. The views expressed herein do not necessarily represent the views of the DOE or the U.S. Government.

Cover Photos by Dennis Schroeder: (clockwise, left to right) NREL 51934, NREL 45897, NREL 42160, NREL 45891, NREL 48097, NREL 46526.

NREL prints on paper that contains recycled content.

Cooperative Research and Development Final Report

Report Date: November 20, 2018

In accordance with requirements set forth in the terms of the CRADA agreement, this document is the final CRADA report, including a list of subject inventions, to be forwarded to the DOE Office of Science and Technical Information as part of the commitment to the public to demonstrate results of federally funded research.

Parties to the Agreement: Radiant Labs, LLC

CRADA Number: CRD-17-00701

CRADA Title: Multifamily Building Stock Modeling

Joint Work Statement Funding Table showing DOE commitment:

Estimated Costs	NREL Shared Resources a/k/a Government In-Kind
Year 1	$200,000.00
TOTALS	$200,000.00

Abstract of CRADA Work:

U.S. multifamily buildings house 35 million households, consuming 4 quads of source energy and spending $48 billion on utility bills every year. Almost all of these households live in urban areas, where cities are taking the lead on setting aggressive energy goals. Cities currently do not have the data and tools necessary to identify and target opportunities to save energy in their building stock.

Building on the open-source, OpenStudio-based ResStock platform, this project will extend the publicly-available ResStock modeling capabilities to the multifamily sector, enabling Radiant Labs, and others, to partner with cities to market and strategically deploy cost-effective, energy efficiency (EE) upgrades directly to high-priority households.

Radiant Labs has been successful in using ResStock's single-family capabilities to provide value to the City of Boulder, Colorado. However, lack of multifamily capabilities in ResStock is a roadblock for Radiant Labs working with New York City, San Francisco, Washington, D.C., and other cities that have expressed significant interest in working with them. In addition, other cities, companies, and utilities can use these open-source capabilities to grow their EE portfolios, thereby multiplying the impact of this project.

This work also enables national-scale analysis of EE potential in multifamily buildings, which is of strong interest to a variety of stakeholders, including the U.S. Department of Energy (DOE) Office of Energy Policy and Systems Analysis (EPSA), DOE Weatherization Assistance Program (WAP), U.S. Department of Housing and Urban Development (HUD), and the Bonneville Power Administration (BPA).

Summary of Research Results:

Task 1. Multifamily Housing Characteristics

This task involved developing a database of multifamily housing stock characteristics using available data from the U.S. Census American Community Survey, American Housing Survey, the U.S. Energy Information Administration (EIA) Residential Energy Consumption Survey, and other data sources.

Characteristics of interest include building geometry, number and distribution of dwelling units, thermal enclosure characteristics (insulation levels, window properties, air leakage), heating/cooling system characteristics, ventilation system characteristics, water heating system characteristics, appliance and plug load usage, and number and behavior of occupants. The new parameters added to represent the low-rise multifamily sector are illustrated in Figure 1.

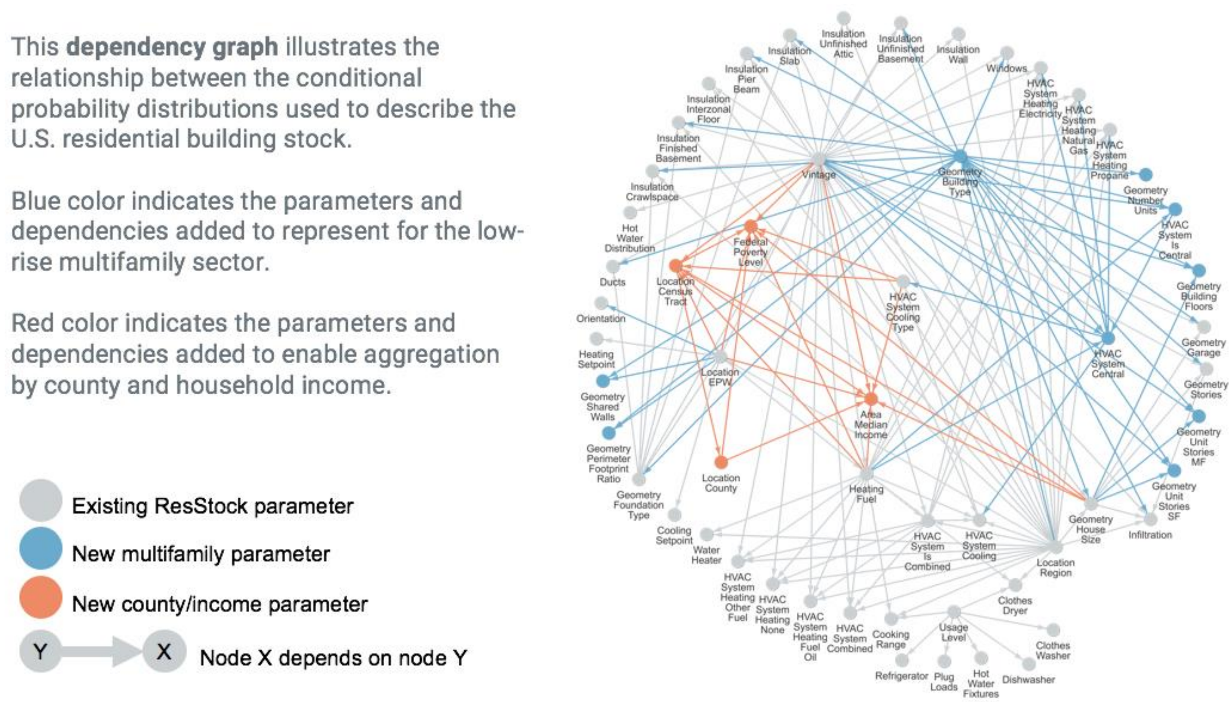


Figure 1. This dependency graph illustrates the relationship between the conditional probability distributions used to describe the U.S. residential building stock. The blue color indicates the parameters and dependencies added to represent for the low-rise multifamily sector.

For each of the characteristics of interest for which data is available, NREL developed scripts that query the data source to develop conditional probability distributions (e.g., probability that a building has a high-efficiency boiler as a function of the building's location and vintage).

Part of Radiant Labs' contribution of in-kind cost share for this project involved Radiant Labs developing scripts that query a OpenStreetMap database of 24 million building footprints to derive conditional probability distributions for several parameters important to modeling the multifamily building stock. See Appendix A for documentation of this process.

The parameters derived from OpenStreetMap include:

- Orientation
- Perimeter footprint Ratio
- Total Wall Area Shared
- Total Wall Area Percentage Shared
- Nearest Neighbor.

Examples of how building footprints were categorized are shown in Figure 2, Figure 3, and Figure 4.

Figure 2. Building geometry input probability distributions were derived from OpenStreetMap data. This example shows building footprints being categorized as detached, middle, or end buildings.

Figure 3. Building geometry input probability distributions were derived from OpenStreetMap data. This example shows building footprints being categorized by footprint aspect ratio.

Figure 4. Building geometry input probability distributions were derived from OpenStreetMap data. This example shows building footprints being categorized by orientation of the footprint's long axis.

Figure 5 shows an example of two types of multifamily building geometry that can be automatically generated using the ResStock scripts that make use of the probability distributions described above.

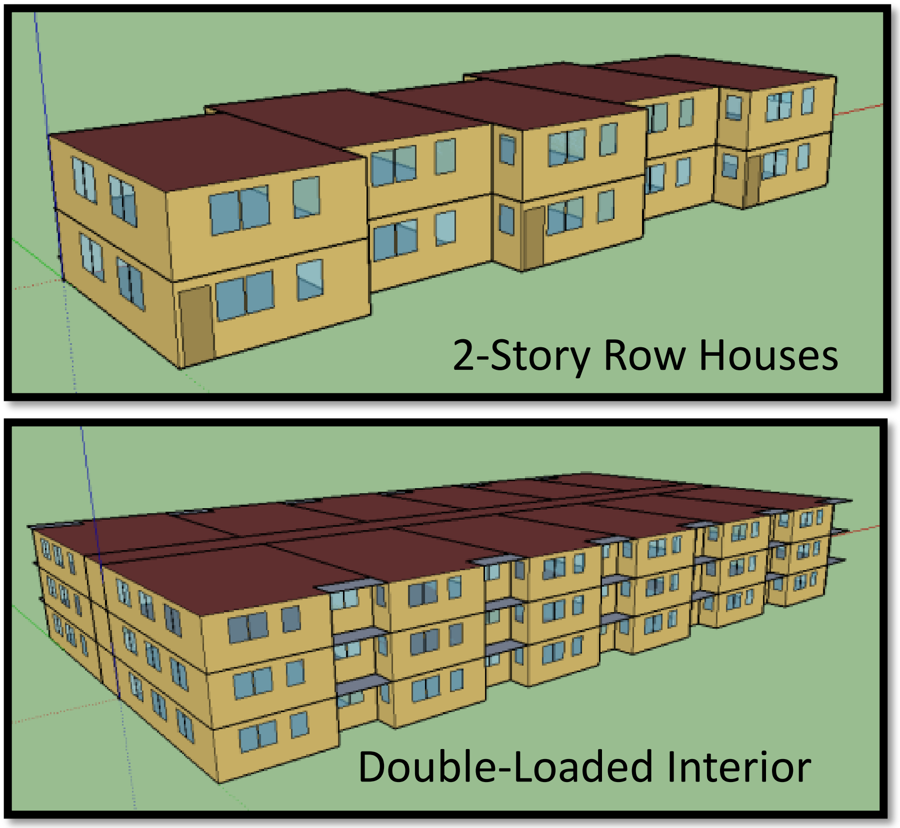


Figure 5. Example of multifamily building geometry that can be automatically generated using ResStock scripts

Task 1 Outcome: Full set of probability distributions developed for multifamily low-rise buildings, published to https://github.com/NREL/OpenStudio-BuildStock/tree/multifamily

Task 2. Multifamily Housing Stock Model Validation
This task involved running Multifamily Housing Stock Model simulations via OpenStudio on cloud computing resources. The simulation results were compared against available data on measured energy consumption of the U.S. multifamily housing stock, from EIA's 2009 Residential Energy Consumption Survey.

Task 2 Outcome: Several iterations of validation comparisons were performed, although the validation process was hampered by stability issues preventing running large simulations of more than 100,000 representative buildings at a time.

Task 3. Enhancements to Multifamily Housing Stock Model Capabilities
This task involved enhancing OpenStudio/EnergyPlus modeling capabilities for some technologies that were not previously available as residential OpenStudio measures. The project team leveraged existing OpenStudio models developed by NREL for commercial buildings analysis, wrapping them for use in the ResStock-Multifamily workflow:

- Fan coils with chiller/boiler
- Fan coils with chiller
- Fan coils with boiler
- Radiators with boiler (hot water or steam)
- Packaged terminal air conditioners (PTACs) with hot water coils served by boiler.

This task included an effort to decrease the runtime of multifamily building simulations through the use of zone/floor multipliers, but discovered a number of barriers which made this effort become not a high priority.

Task 3 Outcome: Multifamily housing stock model capabilities published to https://github.com/NREL/OpenStudio-BEopt

Subject Inventions Listing:

None

ROI #:

None

Responsible Technical Contact at Alliance/National Renewable Energy Laboratory (NREL):

Eric Wilson

Name and Email Address of POC at Company:

Adam Stenftenagel, adam@radiantlabs.co

DOE Program Office:

Office of Energy Efficiency and Renewable Energy (EERE), Building Technologies, SBV Program

Appendix A: OpenStreetMap Footprint Analysis

osm_usa_analysis

May 10, 2018

1 Open Street Map Footprint Analysis

References the calculated values from OSM's building footprints, hosted on Big Query here: https://bigquery.cloud.google.com/table/nrelmultifamily:osm.usa_footprints_calculations?tab=preview

This is in support of NREL's MultiFamily ResStock modeling efforts. If you need access to this dataset, contact jeff@radiantlabs.co

1.0.1 Set up gcloud and authenticate

Install Google's gcloud command-line tools. https://cloud.google.com/bigquery/docs/reference/libraries#clien libraries-install-python

pip install --upgrade google-cloud

pip install --upgrade google-cloud-bigquery[pandas]

Set default project: https://cloud.google.com/sdk/gcloud/reference/config/set

gcloud config set project nrel-multifamily gcloud config get-value project # to confirm

Authentication instructions: https://google-cloud-python.readthedocs.io/en/latest/core/auth.html

gcloud auth application-default login

1.0.2 Imports & config

In [53]:
```
%matplotlib inline
import pandas
as pd

from google.cloud import
bigquery import seaborn as
sns import
matplotlib.pyplot as plt

print('Imported BigQuery lib version: {}. Should be greater than 0.29'.for
```

```
sns.set(color_codes=True)
sns.set(font_scale=2)

plt.rcParams['figure.figsize'] = (20.0, 10.0)

# Set example plots as white to differentiate from the primary plots
sns.set_style("whitegrid")
```

Imported BigQuery lib version: 1.1.0. Should be greater than 0.29

1.0.3 Connect to Big Query Client

```
In [18]: project_id = 'nrel-multifamily' client =
         bigquery.Client(project=project_id) datasets
         = list(client.list_datasets()) if datasets:

             print('Datasets in project {}:'.format(project_id)) for
         dataset in datasets: # API request(s)
         print('\t{}'.format(dataset.dataset_id)) else:

print('{} project does not contain any datasets.'.format(project_id))
```

```
Datasets in project nrel-multifamily:
        osm
```

```
In [19]: dataset_id = 'osm' dataset_ref =
         client.dataset(dataset_id) dataset =
         client.get_dataset(dataset_ref)
```

1.0.4 Example BigQuery query

```
In [ ]: test_query = (

            'SELECT state '

            'FROM `osm.usa_footprints_calculations` '

            'LIMIT 5'

            ) test_query_ref =
        client.query(test_query) rows =
        test_query_ref.result() for row in rows:

            print(row.state)
```

1.0.5 Query BigQuery dataset & import results into Pandas • **Table size**: 11GB

- **Rows**: 24M
- **Requirements**: Greater than 16GB, less than 32 GB RAM. This is to load ~10 columns into Pandas. Fewer columns require less memory

Use the SQL LIMIT clause when testing the results. To get the complete results, normal SQL comments still work.

```
FROM `osm.usa_footprints_calculations`

-- LIMIT 1000
```

In [21]:
```
query_hists = """

        SELECT state, height_ft,
            orientation, aspect_ratio,
            position, perimeter_area_ratio,
            area_volume_ratio,
            total_wall_length_exposed,
            total_wall_length_shared,
            total_wall_length_percentage_s
            hared

        FROM `osm.usa_footprints_calculations`

        -- LIMIT 100

        """ osm_df =
    client.query(query_hists).to_dataframe()
```

In [22]:
```
osm_df.head()
```

Out[22]:
```
      state   height_ft orientation aspect_ratio position \
0  Vermont    43.0    NESW 0.619 Detached
1  Vermont    43.0    NESW 0.438 Detached
2  Nevada    361.0    NESW 0.907   Corner
3  Vermont    49.0    NESW 0.622 Detached
4  Vermont    43.0    NESW 0.898 Detached

 perimeter_area_ratio area_volume_ratio total_wall_length_exposed \
```

0 0.294 0.023 240 1 0.329 0.023 188

2 0.102 0.003 47

3 0.390 0.020 142 4 0.575 0.023 81

total_wall_length_shared total_wall_length_percentage_shared

0 0 0.0

1 0 0.0

2 1889 98.0

3 0 0.0

4 11 12.0

```
In [23]: row_count = len(osm_df)*1
         row_count
```

Out[23]: 24141860

1.0.6 Example: Histogram Plots

The term "clustering" is usually only used with datasets of 2D or greater. For 1D datasets, like a list of numeric values, clustering techniques aren't typically used. Usually something like Kernal

Density Estimation (KDE) is used. https://stackoverflow.com/a/11516590/1884101

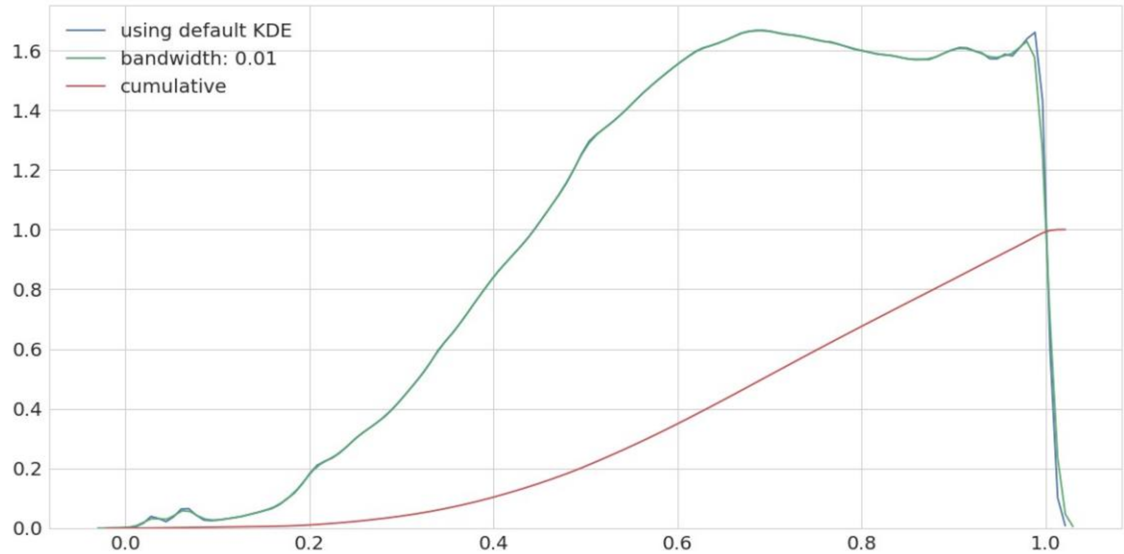
Seaborn distplot has this built in. The bandwidth (bw) parameter sets how tightly the estimation is fit to the data. Seems like the default works well. https://seaborn.pydata.org/tutorial/distributions.html bins: Not specifying the bin count will default using the Freedman-Diaconis rule. Otherwise,

set using bins=20

Using Seaborn (wrapper on top Matplotlib) we can plot a kdeplot with different options:

```
In [24]: sns.kdeplot(osm_df['aspect_ratio'], label="using default KDE")
         sns.kdeplot(osm_df['aspect_ratio'], bw=.01, label="bandwidth: 0.01")
         sns.kdeplot(osm_df['aspect_ratio'], cumulative=True, label="cumulative")
```

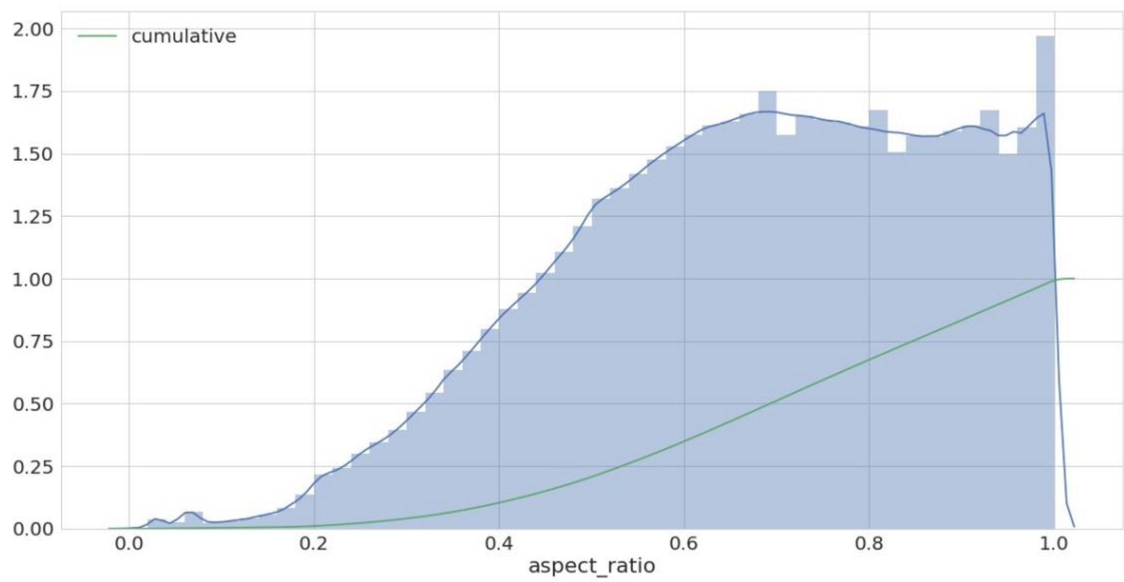
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7efae583df28>



The distplot is similar to kdeplot and also plots the histogram. You can plot the cumulating values on top

In [25]: sns.distplot(osm_df['aspect_ratio']) sns.kdeplot(osm_df['aspect_ratio'], cumulative=**True**, label="cumulative")

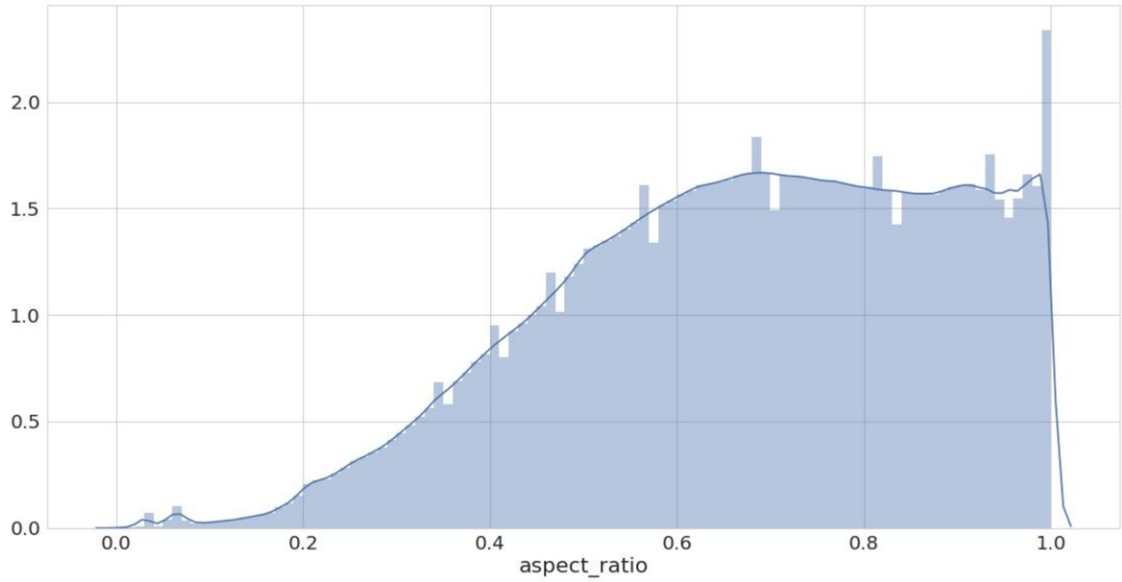
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7efadfd55f98>



You can set the number of bins manually:

In [26]: sns.distplot(osm_df['aspect_ratio'], bins=100)

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7efad7ac0d68>



1.0.7 Example: Quantiles

Histograms use equal-width bins with varying counts (heights) within each bin. But we may want to consider quantiles, which creates bins with the same number of values in it. This makes each bin a different width but the same height.

We can use Pandas qcut for this (https://pandas.pydata.org/pandasdocs/stable/generated/pandas.qcut.html):

In [27]: *# ex_quantiles, ex_bins = pd.qcut(osm_df['aspect_ratio'], 3, retbins=True)*

ex_quantiles, ex_bins = pd.qcut(osm_df['aspect_ratio'], 3, labels=["high" ex_counts = pd.value_counts(ex_quantiles) ex_quantiles.head()

Out[27]: 0 medium

1 high

2 low

3 medium

4 low

Name: aspect_ratio, dtype: category

Categories (3, object): [high < medium < low]

In [28]: print('Bins**n** **{}** **n**'.format(ex_bins)) print('Counts**n** **{}**'.format(pd.value_counts(ex_quantiles)))

Bins

[0.   0.59   0.795 1.   ]

Counts

high   8073399

medium
8059412
low
8009049

Name: aspect_ratio, dtype: int64

2 OSM Distributions

In [29]: *# Set plots to dark to stand out. These are the datasets we really want to*
```
sns.set_style("darkgrid")
```

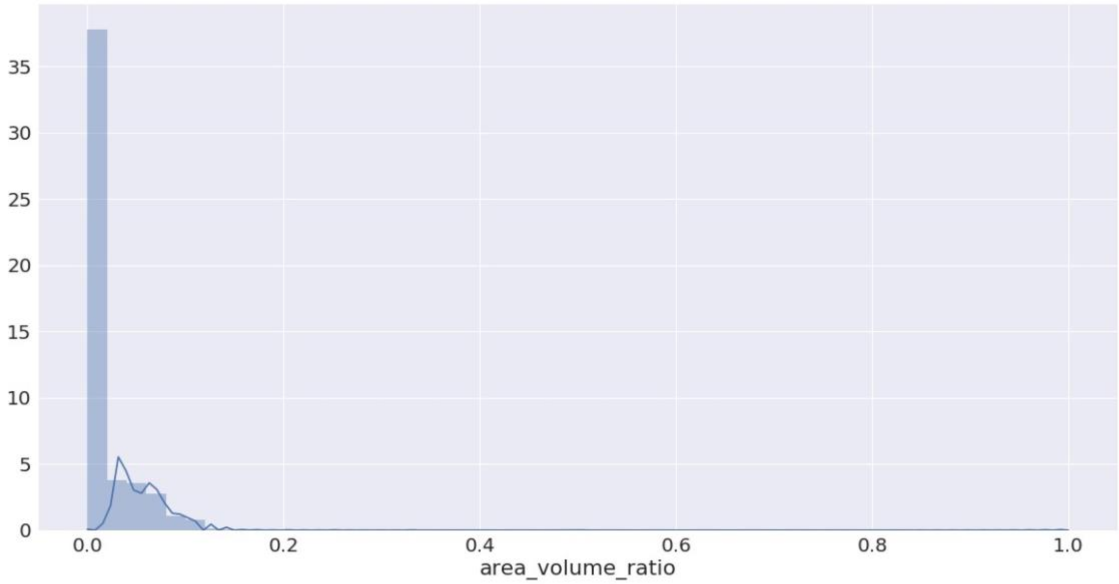
2.1 Perimeter Area Ratio

In [ ]: sns.distplot(osm_df['perimeter_area_ratio'])

2.2 Area Volume Ratio

In [31]: *# Fill NaN with Zeros to get a sense of how much data is missing*
```
sns.distplot(osm_df['area_volume_ratio'].fillna(0))
```

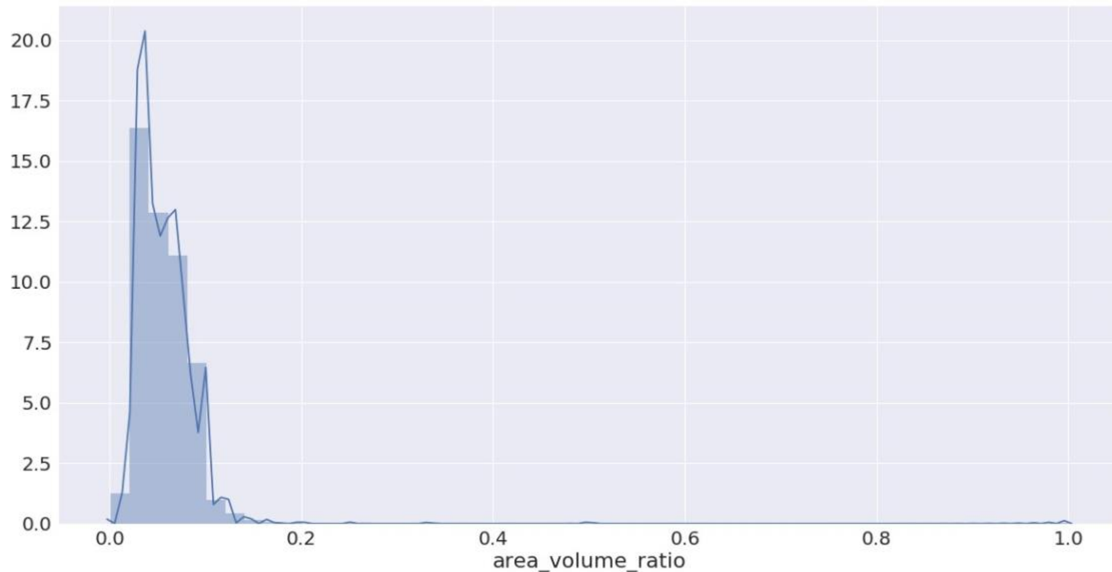
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7eface8ebf28>



In [32]: *# # Plot it while dropping NaNs*

```
sns.distplot(osm_df['area_volume_ratio'].dropna())
```

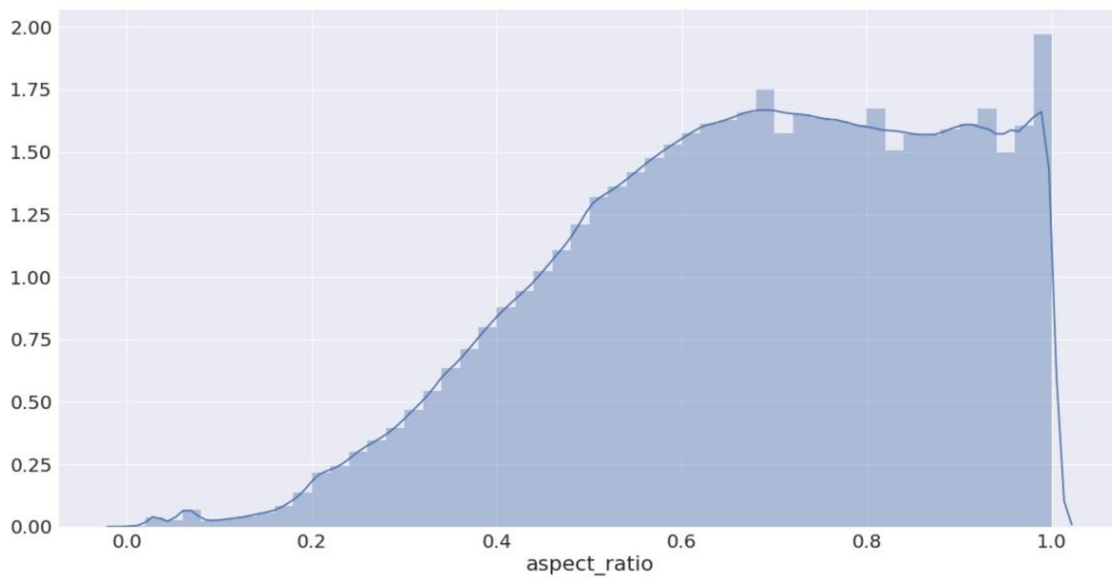
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7eface3010b8>



2.3 Aspect Ratio

In [33]: sns.distplot(osm_df['aspect_ratio'])

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7efad2b57d30>

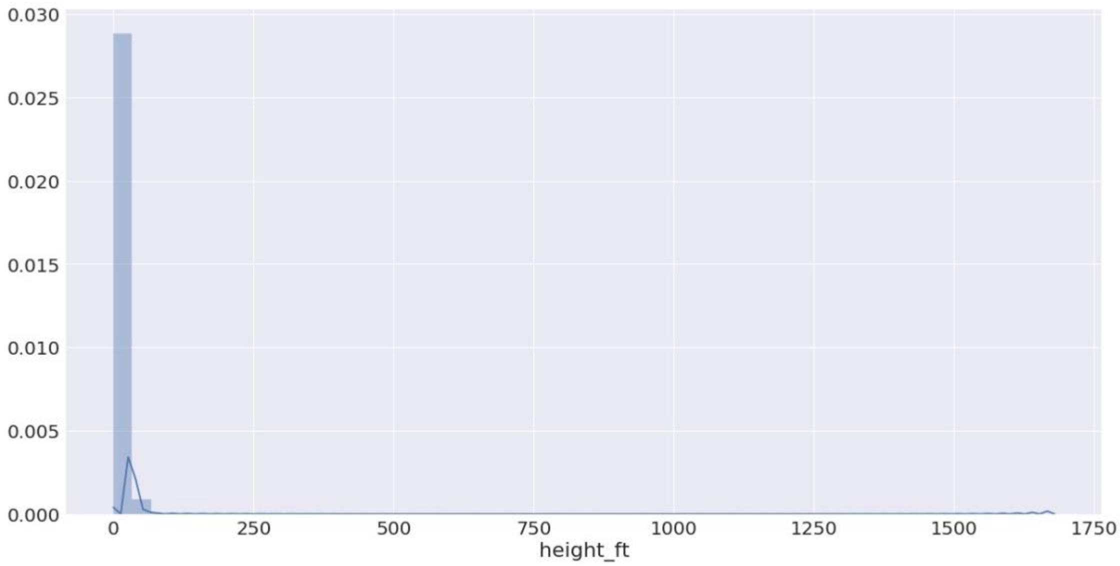


2.4 Building Height

In [34]: *# Fill NaN with Zeros to get a sense of how much data is missing*

```
sns.distplot(osm_df['height_ft'].fillna(0))
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7efaca140080>

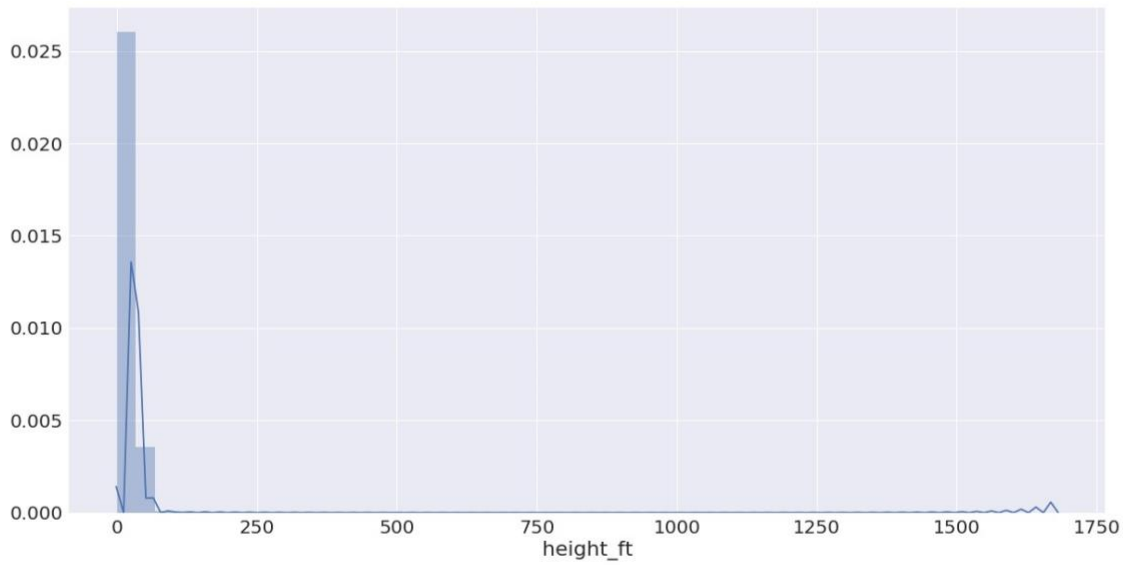


In [

35]: *# Plot it while dropping NaNs*

```
sns.distplot(osm_df['height_ft'].dropna())
```

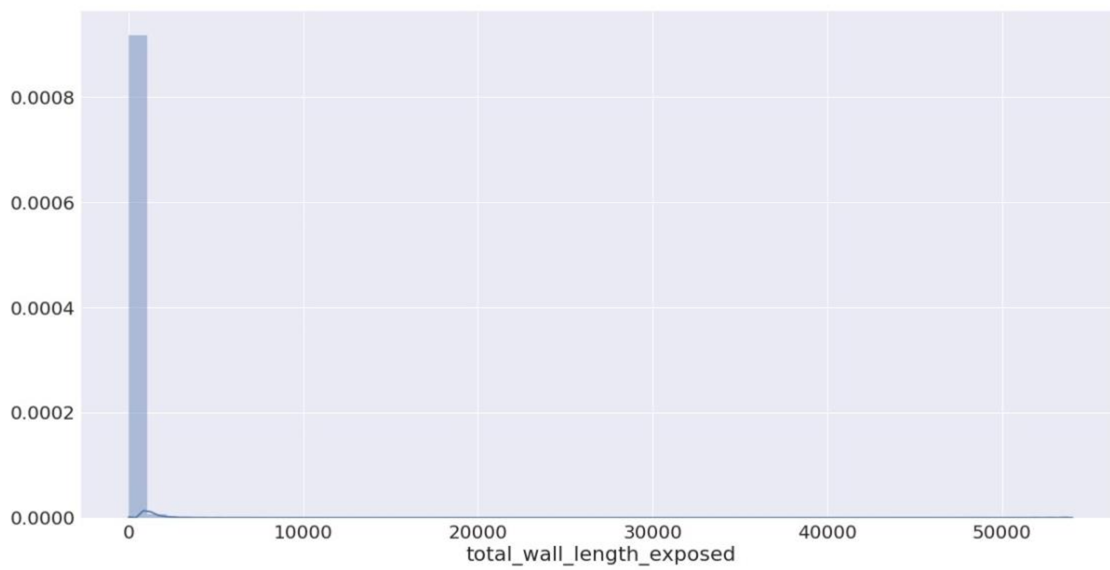
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7efac9ef7dd8>



2.5 Total Wall Length Exposed

In [36]: sns.distplot(osm_df['total_wall_length_exposed'])

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7efac9ccb588>

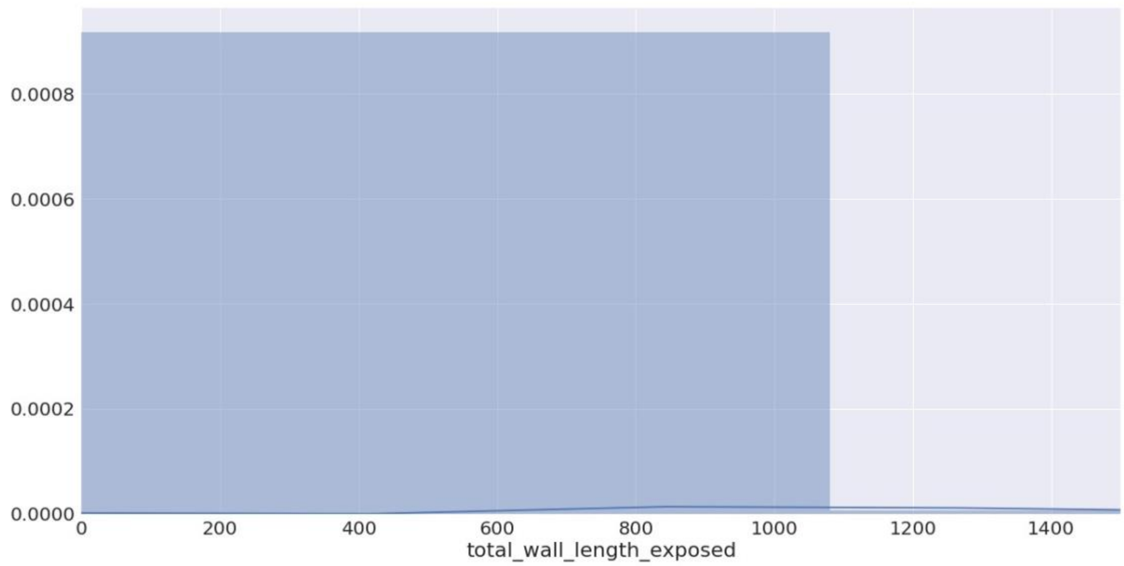


37]:

In [# Zoom in, ignoring the long tail

```
sns.distplot(osm_df['total_wall_length_exposed']).set(xlim=(0, 1500))
```

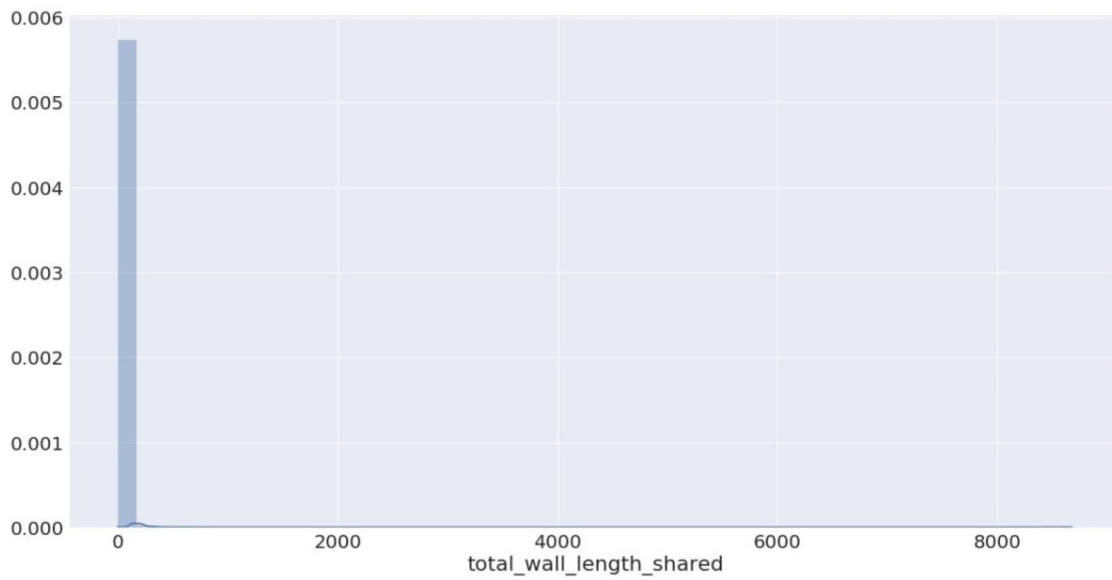
Out[37]: [(0, 1500)]



2.6 Total Wall Length Shared

In [38]: sns.distplot(osm_df['total_wall_length_shared'])

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7efac9837780>

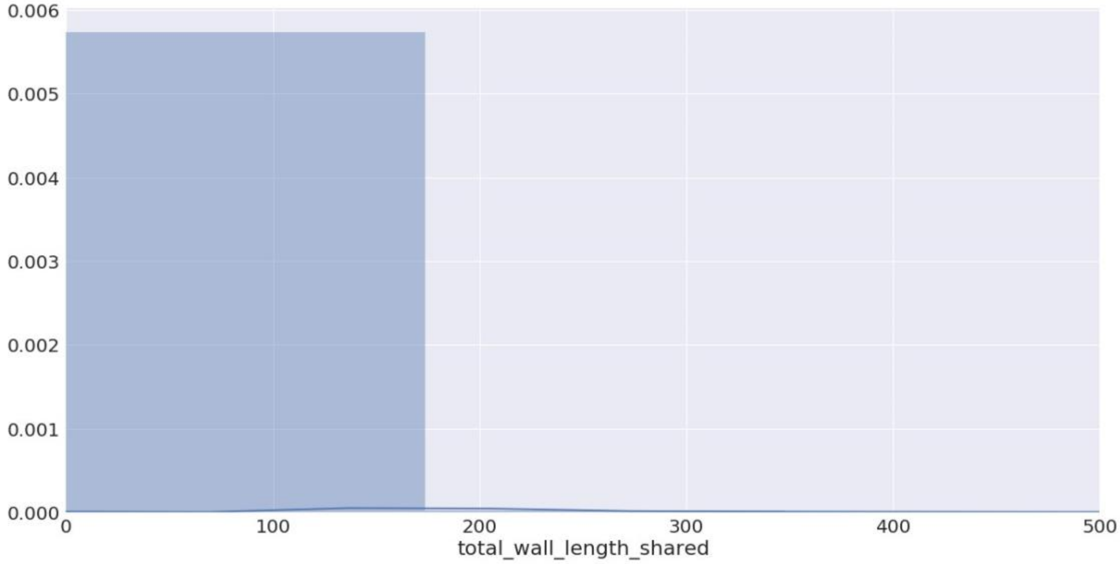


39]:

```
sns.distplot(osm_df['total_wall_length_shared'].set(xlim=(0, 500))
```

In [# Zoom in, ignoring the long tail

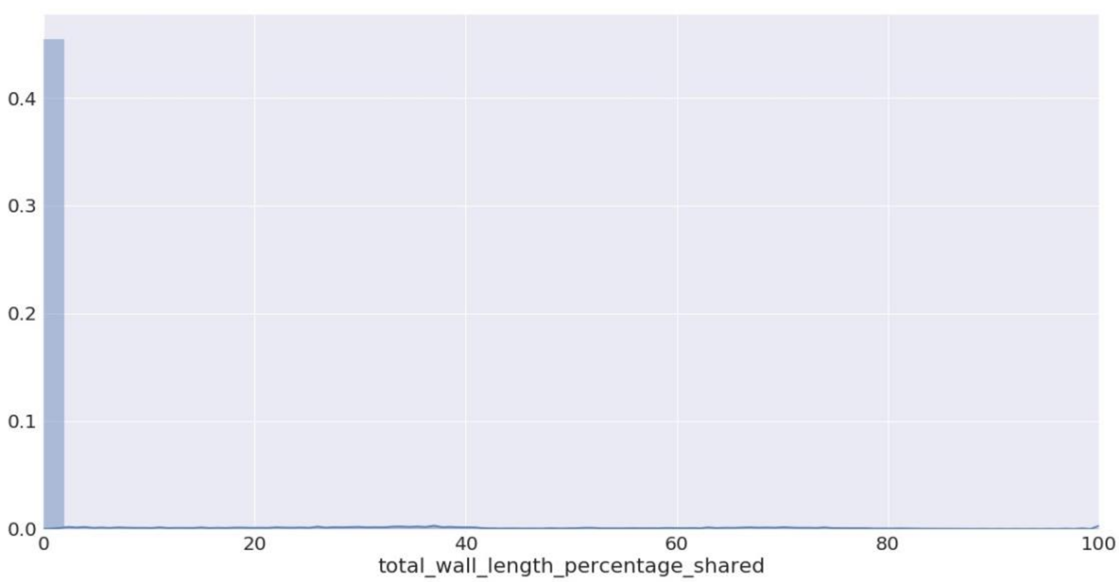
Out[39]: [(0, 500)]



2.7 Total Wall Length Percentage Shared

In [47]: sns.distplot(osm_df['total_wall_length_percentage_shared'].dropna()).set(x

Out[47]: [(0, 100)]

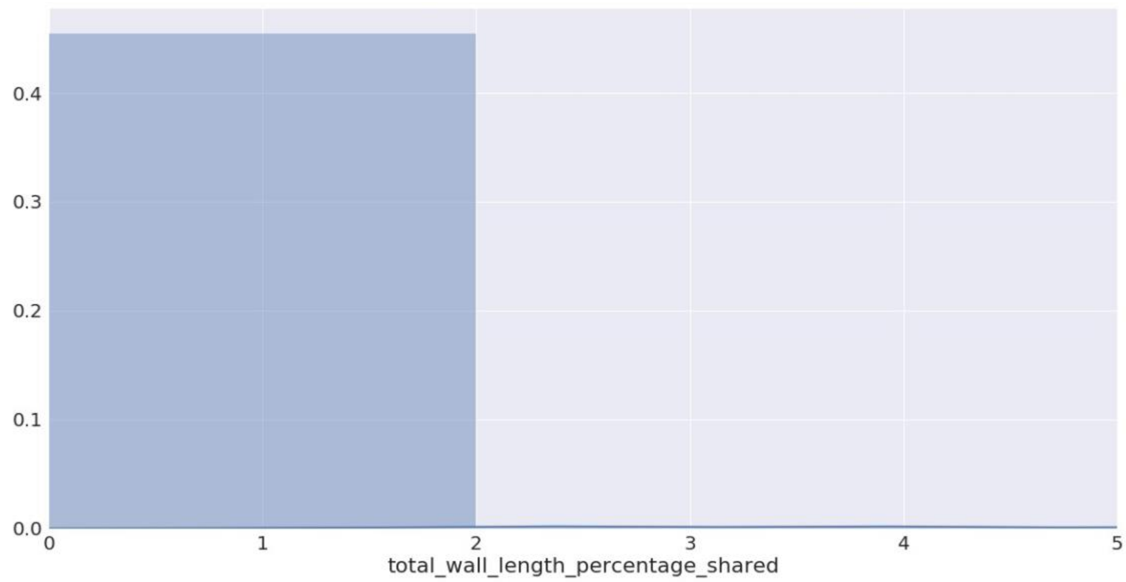


49]:

In [# Zoom in, ignoring the long tail

sns.distplot(osm_df['total_wall_length_percentage_shared'].dropna()).set(x Out[49]:

[(0, 5)]



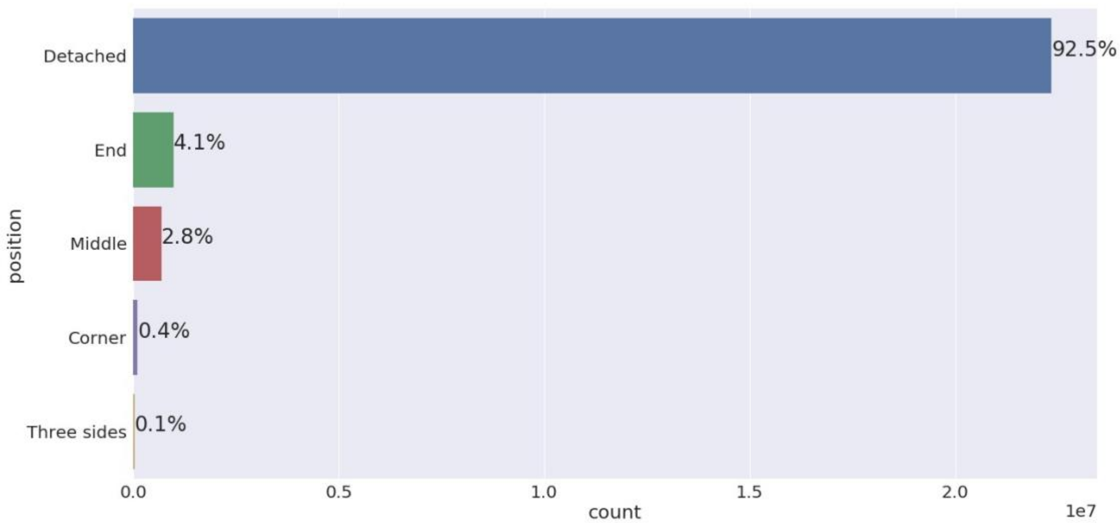
2.8 Position

In [43]: ax = sns.countplot(y='position',

data=osm_df,

order=osm_df['position'].value_counts().index)

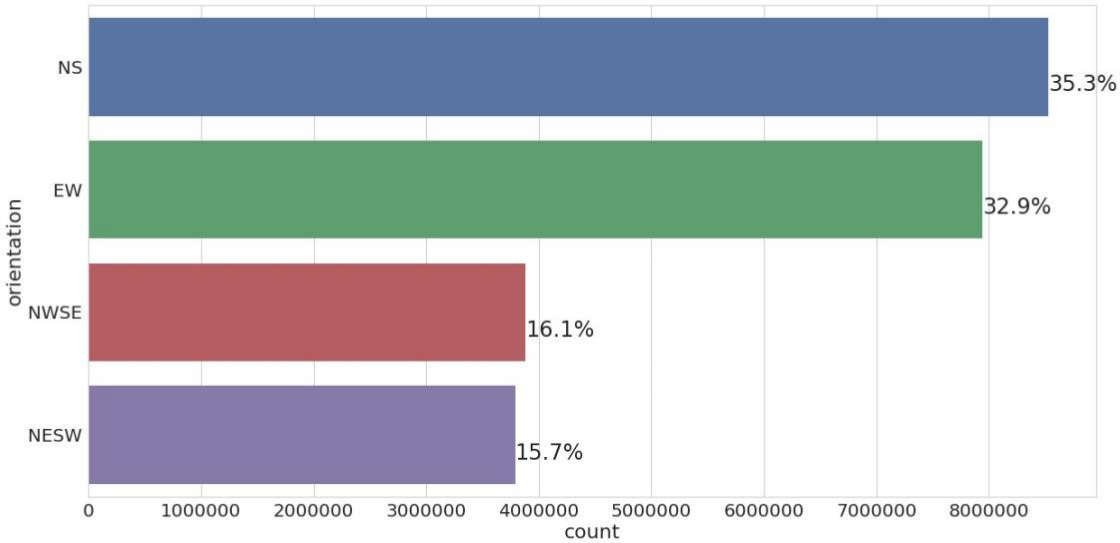
for p **in** ax.patches:



ax.annotate('**{:.1f}**%'.format(100*p.get_width()/row_count), (p.get_widt

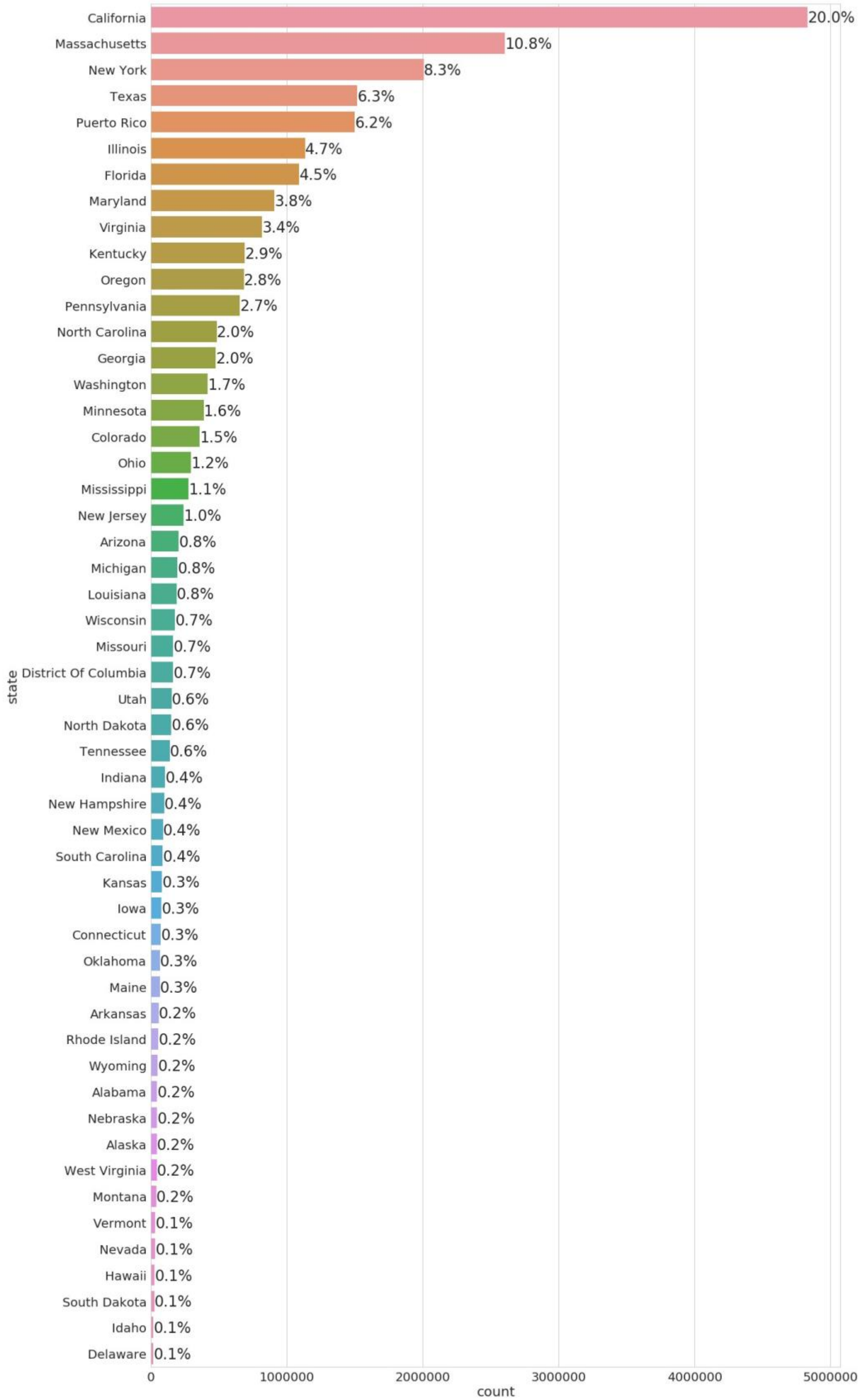
2.9 Orientation

In [54]:
```
ax = sns.countplot(y='orientation', data=osm_df,
                   order=osm_df['orientation'].value_counts().index)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_width()/row_count), (p.get_widt
```



2.10 State

In [59]:
```
# plt.rcParams['figure.figsize'] = (20.0, 30.0)
plt.figure(figsize=(20,40)) ax =
sns.countplot(y='state', data=osm_df,
                   order=osm_df['state'].value_counts().index)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_width()/row_count), (p.get_widt
```


California 20.0%
Massachusetts 10.8%
New York 8.3%
Texas 6.3%
Puerto Rico 6.2%
Illinois 4.7%
Florida 4.5%
Maryland 3.8%
Virginia 3.4%
Kentucky 2.9%
Oregon 2.8%
Pennsylvania 2.7%
North Carolina 2.0%
Georgia 2.0%
Washington 1.7%
Minnesota 1.6%
Colorado 1.5%
Ohio 1.2%
Mississippi 1.1%
New Jersey 1.0%
Arizona 0.8%
Michigan 0.8%
Louisiana 0.8%
Wisconsin 0.7%
Missouri 0.7%
District Of Columbia 0.7%
Utah 0.6%
North Dakota 0.6%
Tennessee 0.6%
Indiana 0.4%
New Hampshire 0.4%
New Mexico 0.4%
South Carolina 0.4%
Kansas 0.3%
Iowa 0.3%
Connecticut 0.3%
Oklahoma 0.3%
Maine 0.3%
Arkansas 0.2%
Rhode Island 0.2%
Wyoming 0.2%
Alabama 0.2%
Nebraska 0.2%
Alaska 0.2%
West Virginia 0.2%
Montana 0.2%
Vermont 0.1%
Nevada 0.1%
Hawaii 0.1%
South Dakota 0.1%
Idaho 0.1%
Delaware 0.1%
state
0
1000000
2000000
3000000
4000000
5000000
count

2.11 Orientation

```
In [58]: ax = sns.countplot(y='orientation', data=osm_df,
                              order=osm_df['orientation'].value_counts().index)
         for p in ax.patches:
                 ax.annotate('{:.1f}%'.format(100*p.get_width()/row_count), (p.get_widt
```